プログラミング基礎'00　＃5

久野　靖*

2000.7.1

0　はじめに

前回の皆様の感想ですが、GUI 部品を用いたソフトウェア作成ができると普段使っているアプリケーションに近付いた感じがするというご感想が複数ありました。確かにそうだとは思います。

一方で、原理的にどういうふうになっているのかは理解するようになったが、自分で一からプログラムを組むのはなかなか難しい、という感想も頂いています。それもその通りだと思います。この授業としては、プログラミングの原理について理解して頂ければ「もっとも最初の」目標は達成したことになると思います。さまざまなプログラムが自由に書けるようになるためには、それなりの時間と訓練が必要なのは当然のことです。

あと、API ドキュメントの見方がよく分からないという感想も多かったです。これはまあ当然だと思いますが、旧来のプログラミングでは「言語の使い方」が中心だったのに対し、これからのプログラミングでは「既にある部品を探してきて応用する」ことも同様に重要になってくる、ということで理解して頂ければと思います。逆に言えば、旧来情報系の学科では必ず教えられてきた、各種のデータ構造や整列、検索などのアルゴリズムについては、ただ「その機能の部品を持って来る」だけで済むので、カジュアルプログラマは「知らなくてもよいこと」になって来ていると思います (それでもプロならやはり知っているべきですが)。

今回の内容ですが、まず今回は資料準備の時間がなかったので、前回の解説がありません。後で別途配布します。そして最終回ということで「落ち穂拾い」で、これまでに扱わなかったファイルの入出力と日本語の扱い、および「アプレットではなく単独で窓を開くプログラム」を取り上げます。その前に、「グラフィクスでなく役に立つクラス」の例を見てみましょう。

注意! 今回の実習はマシン「smp」「smm」「smo」のいずれかの窓の中で行うこと。処理系の違いがちょっと問題になるため。

1　例題: CSV 形式を扱うためのクラス

CSV 形式というのは皆様ご存じの通り、

フィールド 1, フィールド 2,…, フィールド N

のように複数の値をカンマで区切って並べたファイル形式であり、表計算ソフトなどのデータ交換に多く使われている。しかし、それをプログラムで読み込んで処理しようとすると、結構面倒な文字列処理が必要になりそうですね? そこで、次のようなことを考える。自分で CSV を取り扱うためのクラス CSVString を定義し、ファイルから 1 行読み込んだらそれをコンストラクタに渡してオブジェクトを生成する。

```
CSVString csv = new CSVString(line);
```

そうすると、このクラスの中で「フィールドはいくつあるか」「何番目のフィールドの値を取り出す」といった操作が使えるようになる。

*筑波大学大学院経営システム科学専攻

```
int count = csv.getCount();
String second = csv.getField(1);  // 番号は 0 から始まる
```

さらに、特定のフィールドを別の値に置き換えることもできる (その時は新しい CSVString オブジェクトが作られる)。

```
CSVString update = csv.putField(1, "新しい値");
```

さらに指定した位置に新しいフィールドを挿入したり、指定したフィールドを削除することもできる。

```
CSVString update1 = update.insertField(0, "挿入値");
CSVString update2 = update1.removeField(3);
```

これらのことが行えるクラス CSVString を作ってみた。見てみよう。

```
public class CSVString {
  private int[] pos;
  private String str;
```

このクラスでは「もともとの文字列」と、その文字列の何文字目に区切り (カンマ) があったかを覚えておく整数配列で情報を覚えておく。こうした方が「何番目を取り出す」といった操作が速やかに行えるから。コンストラクタではこの配列を初期設定する。

```
  public CSVString(String s) {
    int count = 0;
    for(int i = 0; i < s.length(); ++i) {
      if(s.charAt(i) == ',') { ++count; }
    }
    pos = new int[count+2]; str = s; count = 0; pos[count] = 0;
    for(int i = 0; i <= s.length(); ++i) {
      if(i == s.length() || s.charAt(i) == ',') { ++count; pos[count] = i+1; }
    }
  }
```

まずカンマがいくつあるか数え、それより 2 つぶん多くの要素を格納できる配列を用意する。なぜそうするかというと、フィールドのはじまりと終りまで配列に入れておく方が後で処理がやりやすいから。そして、各要素にはフィールドの開始位置を入れる。上記の理由から「最後のフィールドの次のフィールド」の位置も格納しておく。

```
  public String toString() { return str; }
  public int getCount() { return pos.length - 1; }
  public String getField(int i) {
    if(i < 0 || i >= getCount()) {
      throw new RuntimeException("field# out of range: "+i);
    }
    return str.substring(pos[i], pos[i+1]-1);
  }
```

toString() や getCount() は簡単。getField() は配列に入っている位置をもとに String のメソッド substring() を使って部分文字列を取り出してくる。なお、指定したフィールド番号が範囲外なら「例外を発生する」文を実行している。

```
  public CSVString putField(int i, String s) {
    if(i < 0 || i >= getCount()) {
      throw new RuntimeException("field# out of range: "+i);
    } else if(pos.length <= 2) {
      return new CSVString(s);
    } else if(i == 0) {
      return new CSVString(s + "," +
                   str.substring(pos[i+1], pos[pos.length-1]-1));
    } else if(i == pos.length - 2) {
      return new CSVString(str.substring(pos[0], pos[i]-1) + "," + s);
    } else {
      return new CSVString(str.substring(pos[0], pos[i]-1) + "," + s +
                   "," + str.substring(pos[i+1], pos[pos.length-1]-1));
    }
  }
```

putField() はそのフィールドが唯一のフィールド、先頭/末尾のフィールドの場合を別に扱うので枝分かれが多くなるが、基本的には文字列を「はぎ合わせて」更新された文字列を作り、それを新しい CSVString にして返す。

```
  public CSVString insertField(int i, String s) {
    if(i < 0 || i > getCount()) {
      throw new RuntimeException("field# out of range: "+i);
    } else if(i == getCount()) {
      return new CSVString(str + "," + s);
    } else {
      return putField(i, s + "," + getField(i));
    }
  }
```

insertField() は末尾への追加以外の場合はせっかく作った putField() を利用する。removeField() は putField() とほぼ同様で、置き換える文字列がなくなっただけ。

```
  public CSVString removeField(int i) {
    if(i < 0 || i >= getCount()) {
      throw new RuntimeException("field# out of range: "+i);
    } else if(pos.length <= 2) {
      throw new RuntimeException("single field -- can't remove");
    } else if(i == 0) {
      return new CSVString(str.substring(pos[i+1], pos[pos.length-1]-1));
    } else if(i == pos.length - 2) {
      return new CSVString(str.substring(pos[0], pos[i]-1));
    } else {
      return new CSVString(str.substring(pos[0], pos[i]-1) +
                   "," + str.substring(pos[i+1], pos[pos.length-1]-1));
    }
  }
}
```

結構長いけれど、ちゃんと役に立つ (汎用の) クラスを作ろうとすればこれくらいは当り前。さて、このクラスは特定のプログラムからだけでなく、さまざまなプログラムで利用できる。そこで、このクラスは単独のファ

イルに入れてどこからでも使えるようにしてある。打ち込むのは大変だから、久野のディレクトリからコピーしてくださって結構です。

```
% cp /u1a/kuno/work/CSVString.java CSVString.java
% javac CSVString
```

2 日本語の取り扱い

さて、ここまでわざと延ばしていたが、Java で日本語を扱うことについて学ぼう。Java ではプラットフォーム独立に、世界各国の言語を統一的に扱えるようにするため、プログラム内部では文字コードを Unicode(ISO 10646) と呼ばれるものに統一して扱っている。このため、外部から文字を読み込むとき、および外部に文字を書き出す時には外部のコード (日本語だと JIS、EUC、SJIS のどれかが普通) と内部のコードの間で変換が必要になる。これは実は InputStreamReader(バイト単位での読み込みを文字単位での読み込みに変換する) および OutputStreamWriter(文字単位での書き出しをバイト単位での書き出しに変換する) で行える。

```
... new InputStreamReader(バイトストリーム, "JISAutoDetect") ...
... new OuptutStreamWriter(バイトストリーム, "JIS") ...
```

なお、「JISAutoDetect」は JIS、SJIS、EUC のどれでも自動判別して読み込んでくれる。出力は「JIS」、「SJIS」、「EUCJIS」のどれかを指定する。

もう 1 つ変換が問題になるものがあるのが分かりますか? それは、プログラムの中に文字列 (リテラル) を書くときにどうなるか、ということ。JDK 1.2/1.3 の javac は自動的に変換して読み込みながらコンパイルしてくれようにできるので、この機能が使われるように設定してあります。ただし、Solaris 上のコンパイラは EUC しか扱えないので、次のようにしてください。

```
% toeuc R5Sample1.java
% javac R5Sample1.java
```

これがうまく行かない環境 (FreeBSD など) では、日本語の入ったプログラムファイルと javac に掛けるファイルを分けないといけないのでやや面倒。

```
% native2ascii -encoding JISAutoDetect R5Sample1.jis R5Sample1.java
% javac R5Sample1.java
```

今回は面倒なので Solaris 側のやり方を使ってください。なお、日本語を使っていないファイルはこのような作業はまったく必要ない。

では、上記のやり方で CSV 形式のファイルを読み込み、第 2 フィールドと第 3 フィールドの値を足したものを第 4 フィールドとして追加する、ただし第 1 フィールドが「大阪」の時だけは代りに 0 を追加する、というプログラムを示す。

```
import java.io.*;

public class R5Sample1 {
  public static void main(String[] args) {
    try {
      BufferedReader in = new BufferedReader(new InputStreamReader(
                          new FileInputStream(args[0]), "JISAutoDetect"));
      PrintWriter out = new PrintWriter(
                        new OutputStreamWriter(System.out, "JIS"));
      String line;
      while((line = in.readLine()) != null) {
```

```
          CSVString s1 = new CSVString(line);
          double d1 = (new Double(s1.getField(1))).doubleValue();
          double d2 = (new Double(s1.getField(2))).doubleValue();
          double d3 = d1 + d2;
          if(s1.getField(0).equals("大阪")) { d3 = 0.0; }
          out.println(s1.insertField(3, ""+d3)); out.flush();
        }
      } catch(Exception ex) { System.out.println("error: "+ex); }
    }
  }
```

これを動かす様子を次に示す。まずデータファイルは次の通り。

...

動かすと次のようになる。

```
% javac R5Samle1.java  ←CSVStringは自動的に参照される
% java R5Sample1 r5sample.data  ←ファイル名を指定
...
```

演習 1 上記の例題を打ち込んで動かせ。ただしクラス CSVString のソースファイルはコピーして来たものを使ってよい。例題用のデータファイルも

```
% cp /u1a/kuno/work/r5sample1.data  r5sample1.data
```

によりコピーしてきてよい。内容は次のようになっている (はず)。

```
東京,25,2
大阪,40,-1
名古屋,28,3
福島,10,2
仙台,24,3
浜松,21,-1
福岡,34,4
```

演習 2 プログラムを次のように手直しせよ。

a. 足した値を第 4 フィールドとして追加するのではなく、第 2 フィールドの値を足した値で置き換えるようにする。大阪の特別扱いはやめる。

b. 第 2 フィールドの値の合計、第 3 フィールドの値の合計をそれぞれ計算し、データを全部表示し終った後に

```
合計,xxxx,yyyy
```

のように表示せよ (xxxx、yyyy のところに計算した合計が入る)。

c. 都指名のフィールドが今は先頭フィールドになっているが、これを末尾になるように変換せよ。つまり最後の行なら「34,4, 福岡」になる。

3　自分の窓を開くアプリケーション

さて、ファイルが読み書きできるようになったので、GUI プログラムでこれを利用しよう。しかし実は、アプレットではファイルの読み書きが禁止されている (なぜか分かりますか?)。なので、今回は通常のプログラムからウィンドウを開いて、その中で GUI 操作を行う方法を学ぶ。さらに、そのようなプログラムでは窓の大きさを自由に変更できるので、大きさが変更されても配置がヘンにならないような仕組みであるレイアウトマネージャについても学ぼう。

まず、冒頭部分だが、アプレットは Applet クラスを extends していたが、今度は「単独の窓」なので Frame クラスを extends する。あと必要な GUI 部品をインスタンス変数として用意するのは同じ。

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class R5Sample2 extends Frame {
  List l1 = new List();
  TextField f1 = new TextField();
  Button b1 = new Button("Read");
  Button b2 = new Button("Write");
  Button b3 = new Button("Delete");
  Button b4 = new Button("Insert");
  Button b5 = new Button("Quit");
  Label l2 = new Label("");
```

次に、単独のプログラムなので必ず main() が必要。main() ではこのクラスのオブジェクトを生成し、大きさを設定し、レイアウトマネージャを動かしてから窓を見えるようにする。なお、ウィンドウマネージャの機能で窓が閉じられようとした時にプログラムが終了するよう、「ウィンドウ閉じイベント」を受け止めて処理するようなアダプタを設定しておく。main() の処理は以上。

```
  public static void main(String[] args) {
    Frame f = new R5Sample2();
    f.setSize(400, 480); f.pack(); f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent evt) { System.exit(0); }
    });
  }
```

次に、アプレットなら init() で行っていた処理をここではコンストラクタの中に指定する。

```
  public R5Sample2() {
    setLayout(null);
    add(l1); l1.setBounds(10, 40, 380, 300);
    add(f1); f1.setBounds(10, 360, 380, 30);
    add(b1); b1.setBounds(10, 400, 60, 30);
    add(b2); b2.setBounds(80, 400, 60, 30);
    add(b3); b3.setBounds(150, 400, 60, 30);
    add(b4); b4.setBounds(220, 400, 60, 30);
    add(b5); b5.setBounds(290, 400, 60, 30);
    add(l2); l2.setBounds(10, 430, 380, 30);
    setAction();
  }
```

ボタンの動作指定は見やすさのため、メソッド setAction() に分けている。このメソッドを次に示す。まず Read ボタンでは、ファイルから行を読み込み、List 部品に追加する。

```
private void setAction() {
  b1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      try {
        l2.setText("");
        BufferedReader in = new BufferedReader(new InputStreamReader(
             new FileInputStream(f1.getText()), "JISAutoDetect"));
        l1.removeAll();
        String line;
        while((line = in.readLine()) != null) { l1.add(line); }
        in.close(); f1.setText("");
      } catch(Exception ex) { l2.setText(ex.toString()); }
    }
  });
```

逆に Write では List 部品の各行をファイルに書き出す。

```
  b2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      try {
        l2.setText("");
        PrintWriter out = new PrintWriter(new OutputStreamWriter(
            new FileOutputStream(f1.getText()), "JIS"));
        for(int i = 0; i < l1.getItemCount(); ++i) {
          out.println(l1.getItem(i));
        }
        out.close(); f1.setText("");
      } catch(Exception ex) { l2.setText(ex.toString()); }
    }
  });
```

ボタン 3 と 4 の動作はまだ何もない (練習問題用に用意)。ボタン 5 はプログラムを終了させる。

```
  b3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      try {
        l2.setText("");
        // delete...
      } catch(Exception ex) { l2.setText(ex.toString()); }
    }
  });
  b4.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      try {
        l2.setText("");
        // insert...
      } catch(Exception ex) { l2.setText(ex.toString()); }
```

```
      }
    });
    b5.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        System.exit(0);
      }
    });
  }
}
```

4 レイアウトマネージャ

さて、このプログラムを動かしてみると当然ちゃんと動くが、窓の大きさを変えると大変なさけない状態になる。このように、窓の大きさが変化するような場合はそれに応じて中の部品の大きさも調整してくれるような機能があり、「レイアウトマネージャ」と呼ばれている。レイアウトマネージャには何種類かあるが、ここでは GridBagLayout というクラス名のものを使う。このレイアウトマネージャは、部品を縦横のます目に入れて配置し、ます目の大きさを中に入っている部品の大きさと窓全体の大きさに応じて調整してくれる。

部品を配置するとき、その位置指定情報は GridBagConstraints オブジェクトで指定する。先の例題のコンストラクタ (例題番号を変えたので名前も変えてある) を次のように差し替えた。なお、位置指定の指定方法が結構長々しいので、この部分を layout() という下請けメソッドに分けている。

```
public R5Sample3() {
  GridBagConstraints c = new GridBagConstraints();
  GridBagLayout g = new GridBagLayout(); setLayout(g);
  layout(l1, 1, 1, 5, 1, 1, 100, g, c);
  layout(f1, 1, 2, 5, 1, 1, 1, g, c);
  layout(b1, 1, 3, 1, 1, 1, 1, g, c);
  layout(b2, 2, 3, 1, 1, 1, 1, g, c);
  layout(b3, 3, 3, 1, 1, 1, 1, g, c);
  layout(b4, 4, 3, 1, 1, 1, 1, g, c);
  layout(b5, 5, 3, 1, 1, 1, 1, g, c);
  layout(l2, 1, 4, 5, 1, 1, 1, g, c);
  setAction();
}
private void layout(Component p, int x, int y, int w, int h, int wx, int wy,
                    GridBagLayout g, GridBagConstraints c) {
  c.gridx = x; c.gridy = y; c.gridwidth = w; c.gridheight = h;
  c.weightx = wx; c.weighty = wy; c.fill = GridBagConstraints.BOTH;
  g.setConstraints(p, c); add(p);
}
```

layout() は「どのセル」「何セルぶんぶち抜き」「伸び指数」をパラメタから設定し、さらに「部品をセル一杯にのばす」という指定を行ってから部品を add() する。こちらを使えば、窓の大きさを変えても大丈夫である。

演習 3 この 2 つの例題プログラムのソースは

```
% cp /u1a/kuno/work/R5Sample2.java R5Sample2.java
% cp /u1a/kuno/work/R5Sample3.java R5Sample3.java
```

でコピーしてきてよい。どちらか好きな方をとってきて、このまま動かせ。動いたら次のことを行え。

a. ボタン 3 に「リストで選択されている行を削除する」という動作をつけよ。
b. ボタン 4 に「テキスト入力欄に打ち込んだ内容をリストの選択されている行のすぐ上に追加する」という動作をつけよ。
c. リストのある行をテキスト入力欄に持ってきて、それを編集し、また同じ場所に戻すような機能を追加してみよ。もちろん、ボタンを増やす必要がある。

5 おまけ: ファイル保存可能な図形記述アプリケーション

最後の例題として、図形の生成指定をファイルに書いておいてそれを読み込み、「実行」することでさまざまな図形や動きが画面に配置できる、という例題を作成してみた。全部で 300 行近くあるが、これまでやった内容の集大成だと思って読んでみて頂きたい。冒頭部分はさっきまでのと同じだが、図形を表示するキャンバスを java.awt.Canvas のサブクラス MyCanvas として用意し、それを List の横に配置している。またボタンがだいぶ増えている。

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class R5Sample4 extends Frame {
  List l1 = new List();
  MyCanvas c1 = new MyCanvas();
  TextField f1 = new TextField();
  Button b1 = new Button("Read");
  Button b2 = new Button("Write");
  Button b3 = new Button("Delete");
  Button b4 = new Button("Insert");
  Button b5 = new Button("Quit");
  Button b6 = new Button("View");
  Button b7 = new Button("Start");
  Button b8 = new Button("Stop");
  Label l2 = new Label("");
```

main() や部品の配置についてはこれまでと同様。

```
  public static void main(String[] args) {
    Frame f = new R5Sample4();
    f.setSize(400, 400); f.pack(); f.setVisible(true);
    f.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent evt) { System.exit(0); }
    });
  }
  public R5Sample4() {
    GridBagConstraints c = new GridBagConstraints();
    GridBagLayout g = new GridBagLayout(); setLayout(g);
    layout(l1, 1, 1, 4, 1, 20, 100, g, c);
    layout(c1, 5, 1, 5, 1, 80, 100, g, c);
    layout(f1, 1, 2, 8, 1, 1, 1, g, c);
```

```
    layout(b1, 1, 3, 1, 1, 1, 1, g, c);
    layout(b2, 2, 3, 1, 1, 1, 1, g, c);
    layout(b3, 3, 3, 1, 1, 1, 1, g, c);
    layout(b4, 4, 3, 1, 1, 1, 1, g, c);
    layout(b5, 5, 3, 1, 1, 1, 1, g, c);
    layout(b6, 6, 3, 1, 1, 1, 1, g, c);
    layout(b7, 7, 3, 1, 1, 1, 1, g, c);
    layout(b8, 8, 3, 1, 1, 1, 1, g, c);
    layout(l2, 1, 4, 8, 1, 1, 1, g, c);
    setAction();
  }
  private void layout(Component p, int x, int y, int w, int h, int wx, int wy,
                  GridBagLayout g, GridBagConstraints c) {
    c.gridx = x; c.gridy = y; c.gridwidth = w; c.gridheight = h;
    c.weightx = wx; c.weighty = wy; c.fill = GridBagConstraints.BOTH;
    g.setConstraints(p, c); add(p);
  }
```

動作であるが、ファイルの読み書きは先の例題と同様。

```
  private void setAction() {
    b1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        try {
          l2.setText("");
          BufferedReader in = new BufferedReader(new InputStreamReader(
               new FileInputStream(f1.getText()), "JISAutoDetect"));
          l1.removeAll();
          String line;
          while((line = in.readLine()) != null) { l1.add(line); }
          in.close(); f1.setText("");
        } catch(Exception ex) { l2.setText(ex.toString()); }
      }
    });
    b2.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        try {
          l2.setText("");
          PrintWriter out = new PrintWriter(new OutputStreamWriter(
              new FileOutputStream(f1.getText()), "JIS"));
          for(int i = 0; i < l1.getItemCount(); ++i) {
            out.println(l1.getItem(i));
          }
          out.close(); f1.setText("");
        } catch(Exception ex) { l2.setText(ex.toString()); }
      }
    });
```

行の挿入と削除は一応こういう風にやってみました (好みによってもっと違う形でもよいけど)。終了は同じ。

```
    b3.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        try {
          l2.setText("");
          f1.setText(l1.getSelectedItem());
          l1.remove(l1.getSelectedIndex());
        } catch(Exception ex) { l2.setText(ex.toString()); }
      }
    });
    b4.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        try {
          l2.setText("");
          if(l1.getSelectedIndex() < 0) {
            l1.add(f1.getText());
          } else {
            l1.add(f1.getText(), l1.getSelectedIndex());
          }
          f1.setText("");
        } catch(Exception ex) { l2.setText(ex.toString()); }
      }
    });
    b5.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        System.exit(0);
      }
    });
```

さて、「View」ボタンが押されたときは、List に入っている各行を順に取り出し、CSVString でフィールドに分ける。最初のフィールドがコマンドであり、コマンドに応じて処理を振り分けている。

- 色,R,G,B — これから作成する図形の色を指定する。
- 円,r — 半径を指定して円を作成。
- 矩形,w,h — 幅と高さを指定して矩形を作成。
- 回転多角形,n,r,v — 変の数、半径、角速度を指定して回転する多角形を作成。

図形を作成した後で、次のいずれかを使って図形を画面に「配置」する。

- 静止配置,x,y — 決まった位置に配置する (動かない)。
- 円周運動,x,y,r,t,v — (x, y) を中心とし、半径 r、最初の角度 t、角速度 v に従って円周運動させる。
- 直線運動,x,y,dx,dy,f — (x, y) から $(x + dx, y + dy)$ まで直線状に移動し、終点に来ると最初の位置にワープすることを、f 秒に 1 回の割合で行なう。

たとえば次のような感じで記述する。

```
色,255,0,0
円,20
静止配置,100,100
色,0,255,0
```

```
円,30
円周運動,100,120,80,0,1
色,0,0,255
回転多角形,3,30,3
直線運動,120,240,-10,-200,5
```

これを処理する部分が全部ボタンの動作として記述されている。参照する各クラスは後で出て来る。

```
b6.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      c1.reset();
      Color c = Color.black;
      Figure f = new Circle(Color.black, 0.0, 0.0, 10.0);
      for(int i = 0; i < l1.getItemCount(); ++i) {
        CSVString csv = new CSVString(l1.getItem(i)); l1.select(i);
        String cmd = csv.getField(0);
        if(cmd.equals("色")) {
          c = new Color(new Integer(csv.getField(1)).intValue(),
                        new Integer(csv.getField(2)).intValue(),
                        new Integer(csv.getField(3)).intValue());
        } else if(cmd.equals("円")) {
          f = new Circle(c, 0.0, 0.0,
                         new Double(csv.getField(1)).doubleValue());
        } else if(cmd.equals("矩形")) {
          f = new Rect(c, 0.0, 0.0,
                       new Double(csv.getField(1)).doubleValue(),
                       new Double(csv.getField(2)).doubleValue());
        } else if(cmd.equals("回転多角形")) {
          f = new RegularPolygon(c,
                       new Integer(csv.getField(1)).intValue(),
                       0.0, 0.0,
                       new Double(csv.getField(2)).doubleValue(),
                       new Double(csv.getField(3)).doubleValue());
        } else if(cmd.equals("静止配置")) {
          c1.addFigure(new StillMove(f,
                         new Double(csv.getField(1)).doubleValue(),
                         new Double(csv.getField(2)).doubleValue()));
        } else if(cmd.equals("円周運動")) {
          c1.addFigure(new CircleMove(f,
                         new Double(csv.getField(1)).doubleValue(),
                         new Double(csv.getField(2)).doubleValue(),
                         new Double(csv.getField(3)).doubleValue(),
                         new Double(csv.getField(4)).doubleValue(),
                         new Double(csv.getField(5)).doubleValue()));
        } else if(cmd.equals("直線運動")) {
          c1.addFigure(new LinearMove(f,
                         new Double(csv.getField(1)).doubleValue(),
                         new Double(csv.getField(2)).doubleValue(),
```

```
                            new Double(csv.getField(3)).doubleValue(),
                            new Double(csv.getField(4)).doubleValue(),
                            new Double(csv.getField(5)).doubleValue()));
              } else {
                throw new RuntimeException("unknown command: "+cmd);
              }
            }
            c1.repaint();
          } catch(Exception ex) { l2.setText(ex.toString()); }
        }
      });
```

アニメーションの開始と終了は MyCanvas オブジェクトのメソッドを呼ぶだけ。

```
      b7.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
          try {
            c1.start();
          } catch(Exception ex) { l2.setText(ex.toString()); }
        }
      });
      b8.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
          try {
            c1.stop();
          } catch(Exception ex) { l2.setText(ex.toString()); }
        }
      });
    }
```

さて、図形を統一的に扱うためのインタフェースであるが、今回は図形の位置を後から設定できるようにメソッド moveTo() が追加してある。

```
    interface Figure {
      public void addTime(double dt);
      public void moveTo(double x, double y);
      public void draw(Graphics g);
    }
```

キャンバスクラスは次の通り。この部分はアプレット本体にちょっと似ている。start() が呼ばれると Thread クラスの無名のサブクラスのインスタンスを生成し、その start() を呼び出すのでこのクラス内で定義されている run() が実行開始される。その内容は一定時間待ちしては repaint() を呼ぶというもの。stop() は running を false にするだけ。

```
    class MyCanvas extends Canvas {
      Figure[] figs = new Figure[100];
      int count = 0;
      boolean running = false;
      long basetime = 0l;
      public void reset() { count = 0; running = false; }
```

```
  public void addFigure(Figure f) {
    if(count+1 < figs.length) { figs[count] = f; ++count; }
  }
  public void paint(Graphics g) {
    double dt = 0.001 * (System.currentTimeMillis() - basetime);
    basetime = System.currentTimeMillis();
    for(int i = 0; i < count; ++i) {
      figs[i].addTime(dt); figs[i].draw(g);
    }
  }
  public void start() {
    (new Thread() {
      public void run() {
        basetime = System.currentTimeMillis();
        running = true;
        while(running) {
          try { sleep(100); } catch(Exception ex) { }
          repaint();
        }
      }
    }).start();
  }
  public void stop() { running = false; }
}
```

さて、今回は継承 (extends) を活用する例を示したいので、すべての図形の基本となる「基本図形」を用意する。このクラスは特定の形を持たない「抽象クラス」で、メソッド draw() は定義していないし、インスタンスも生成できない。

```
abstract class BaseFigure implements Figure {
  double cx, cy, time;
  public BaseFigure(double x, double y) { cx = x; cy = y; }
  public void moveTo(double x, double y) { cx = x; cy = y; }
  public void addTime(double dt) { time = time + dt; }
}
```

このクラスを継承して Cirlce を作る。Circle のコンストラクタの冒頭では親クラスである BaseFigure のコンストラクタを「`super(...)` という書き方で呼び出す。

```
class Circle extends BaseFigure {
  Color col;
  double rad;
  public Circle(Color c, double x, double y, double r) {
    super(x, y); col = c; rad = r;
  }
  public void draw(Graphics g) {
    g.setColor(col);
    g.fillOval((int)cx, (int)cy, (int)(2.0*rad), (int)(2.0*rad));
  }
}
```

矩形もほとんど同様。

```
class Rect extends BaseFigure {
  Color col;
  double width, height;
  public Rect(Color c, double x, double y, double w, double h) {
    super(x, y); col = c; width = w; height = h;
  }
  public void draw(Graphics g) {
    g.setColor(col);
    g.fillOval((int)cx, (int)cy, (int)width, (int)height);
  }
}
```

回転する多角形も同様。

```
class RegularPolygon extends BaseFigure {
  Color col;
  double rad, vtheta;
  int num;
  int[] px, py;
  public RegularPolygon(Color c, int n, double x, double y, double r,
                        double v) {
    super(x, y); col = c; num = n; rad = r; vtheta = v;
    px = new int[n]; py = new int[n];
  }
  public void draw(Graphics g) {
    for(int i = 0; i < num; ++i) {
      double theta = time*vtheta + (2.0 * Math.PI) * i / num;
      px[i] = (int)(cx + rad * Math.cos(theta));
      py[i] = (int)(cy + rad * Math.sin(theta));
    }
    g.setColor(col); g.fillPolygon(px, py, num);
  }
}
```

さて、こんどは「動き」の方のクラスに移る。「静止した動き」のクラスはやはり BaseFigure のサブクラスだが、インスタンス変数 fig を追加してあり、ここに「動かされる図形オブジェクト」を保持する。時間を進めるときもこのオブジェクトの時間と図形オブジェクトの時間をともに進める。

```
class StillMove extends BaseFigure {
  Figure fig;
  public StillMove(Figure f, double x, double y) { super(x, y); fig = f; }
  public void addTime(double dt) { super.addTime(dt); fig.addTime(dt); }
  public void draw(Graphics g) { fig.moveTo(cx, cy); fig.draw(g); }
}
```

円周状の運動は前にやったのと同様だが、親クラスで定義したままで済むメソッドはそのままにしてある。

```
class CircleMove extends StillMove {
  double rad, theta, vtheta;
```

```
    public CircleMove(Figure f, double x, double y, double r,
                      double t, double v) {
      super(f, x, y); rad = r; theta = t; vtheta = v;
    }
    public void addTime(double dt) {
      theta = theta + vtheta*dt; fig.addTime(dt);
    }
    public void draw(Graphics g) {
      fig.moveTo(cx + rad*Math.cos(theta), cy + rad*Math.sin(theta));
      fig.draw(g);
    }
  }
```

直線状の運動も同様 (こちらは addTime() は継承したもののままでよい)。

```
  class LinearMove extends StillMove {
    double dx, dy, freq;
    public LinearMove(Figure f, double x, double y,
                      double dx1, double dy1, double fr) {
      super(f, x, y); dx = dx1; dy = dy1; freq = fr;
    }
    public void draw(Graphics g) {
      fig.moveTo(cx + dx*(time%freq)/freq, cy + dy*(time%freq)/freq);
      fig.draw(g);
    }
  }
}
```

どうですか、かなり長いけれど、クラスに分けてあれば結構読めるようになっているのではないでしょうか? まあ「作る」方はまた別の問題ですが…

演習 4 このプログラムのソース、および上の図形記述は

```
% cp /u1a/kuno/work/R5Sample4.java R5Sample4.java
% cp /u1a/kuno/work/r5sample4.data r5sample4.data
```

によりコピーして来られる。コピーしてきてそのまま動かせ。動いたら図形記述を追加してみよ (プログラムを動かした状態で追加できる)。納得したら、新しい図形、または新しい動きを追加してみよ (当然、それに対応するコマンドも増やす必要がある)。

A 成績評価について

この科目の成績評価ですが、毎回のレポート点と最終レポート点を合わせてつけます。ふつうに出ている人は「C」以上は確定です。だたし出ているだけで「A」は差し上げられません。なぜなら、「プログラムが書けるように」ならなければこの科目をクリアしたとは本来言えないからです。「書ける」という意味もさまざまですが、ここでは次のようなレベルを考えます。

0. 例題を打ち込んでそのまま動かせる。
1. 例題をちょっとだけ直せば済むような課題プログラムならできる。

2. 疑似コードが与えられてその通りのプログラムが作れる。

ここまででは「書ける」とは言えないでしょう。あくまでも、自分でゼロから作れてはじめて「プログラムが書ける」と言えるわけです。これを「レベル 3」と呼ぶことにします。「レベル 3」をさらに次のように (プログラムの「質」によって) 細分化します。

3A. 1 行ないし複数行入力して、その行をもとに指定された加工を行い、出力するようなプログラムが作れる。

3B. 複数行の入力を累積して行って (または配列に入れて処理するなどして) 結果を求めるようなプログラムが作れる。

3C. 複数のコマンドやボタン入力に応じてファイルの読み書きや 3B に相当する動作を行い、全体として役に立つプログラムが作れる。

ここで 3C まで来ていれば結構「役に立つ」ものが作れるようになったものと思います。しかし、プログラムの処理が複雑なものになってくると、さらに次のような技能が必要になって来ます。

4A. プログラムの動作を実現するのに必要な「下請けの」クラスを自分で判断して作ることができる。

4B. 単独のクラスではなく、自分が必要とする複数のクラス群の機能一式を判断して設計し、作ることができる。

4C. そのようにして作るクラス群の構造や相互関係に配慮し、将来作業内容が変化した場合も柔軟に拡張/対応できるように設計できる。

私の希望としては、できれば皆様に 4A くらいまでマスターして頂ければと思っていましたが、皆様の行動を観察していると、全員は無理としても何人かはできているだろうと思います。あと、これらとはまったく直交した技能として次のものがあります。

X. 自分のやりたいことと自分のプログラミングの腕前をすり合わせて、自分が作れる範囲内で欲しい機能を実現するプログラムの機能/構成を設計し、実現する。

実はある意味ではこれが一番重要なのですが、この 5 回の講義ではそれを直接扱うのは無理でした (ある程度プログラムができるようにならないと、こういうことを考えるゆとりがない)。この部分は今後皆様が自分の必要に応じてプログラムを作りながら習得して行かれることを希望します。たとえばレベルが 3A でも、その範囲内で自分に役に立つプログラムはちゃんと作れます。

B 最終課題「5B」

さて、課題ですが、「X」は除いた上の各レベルに相当すると思われる課題を以下に複数用意しました。これらの中から「自分のできる範囲で」なるべく上位レベルの課題を 1 つまたは 2 つ選んでプログラムを作成し、報告してもらいます (2 つ提出した場合は「良い方」で評価します)。それをもとに、到達したレベルに応じて「A」「B」「C」のいずれかの評価をつけさせて頂きます。期限は成績報告期限があるのでいつもより 1 日少なくて 6 日間、7/7(金) 一杯まで (本当に厳守!) になります。提出物はいつもと違って次の形を取ること。

(0) すべて A4 版の用紙を用い、ホチキス等で綴じること。

(1) 1 枚目を表紙とし、5B、学籍番号、氏名、日付、選択した問題番号 (1 つまたは 2 つ) のみをはっきり記入する。

(2) やった問題それぞれについて、リスティングと簡単な説明。

(3) 最後に全体を通じての感想。

いつも通り久野のメールボックスに、ただし 7/7(金) 一杯までに、提出してください。では頑張って!

3A-1 2 つの数値 x、y を入力し、x の y 乗を計算するプログラムを作成せよ。y は正の整数であるものとしてよい。x として 0 が入力されるまで何回でも計算できることが望ましい。

3A-2 文字列 s を入力し、次のような感じで「三角形」を出力するプログラムを作成せよ。文字列として空文字列が入力されるまで (または^D が押されるまで) 何回でも計算できることが望ましい。

```
String> abcde
abcde
abcd
abc
ab
a
String>
```

3B-1 複数の数値を順次入力していき、最後に「0」を入力すると終るものとする。入力した数値の合計と (できれば) 平均を出力するプログラムを作成せよ。

3B-2 上と同様に数値を入力する。入力した数値のうちで、最も平均に近い (平均との絶対値の差が最小の) 数値を (できればそれが何番目のものだったかも併せて) 出力するプログラムを作成せよ。ただし数値の個数は 100 未満と仮定してよい。

3C-1 次のような GUI プログラム (アプレットでも独立プログラムでもよい) を作成せよ。4 つの Label 部品の間に 3 つの Choice 部品が

```
L C L C L C L
```

のようにはさまった画面を持つものとする。Choice 部品は「+- ×÷」の 4 つの選択になっているものとする。「問題」ボタンを押すと 4 つのラベルにランダムに (ヒント: Math.random() を使う)0〜9 の数値が入る。Choide で 3 箇所の演算を選択してから「計算」ボタンを押すと式の値が計算されて結果が表示される。ただそれだけ。計算はちゃんと乗除を先にやるようにしてもよいし、大変なら全部左から順にやってもよい。

3C-2 List 部品が「X」「A」「B」の 3 つある画面の、次のようなプログラムを作成せよ。ファイルを指定して List 部品 X に読み込む。次に、その上で行を選択してから「A」「B」と書かれたボタンのいずれかを押すと選択された行が消え、リスト部品 A または B の末尾に追加される。最後にファイル名を指定して A、B をそれぞれ好きなファイルに書き込むことができる。

4A-1 複素数電卓を作れ。アプレットでも独立プログラムでもよく、画面設計は自由。加算、減算、乗算は必要。除算は分からなければ省略してもよい。ただし複素数をクラスとして定義すること。

4A-2 さまざまな 2 次関数のグラフを「同時に」表示するプログラムを作れ。アプレットでも独立プログラムでもよい。ax^2+bx+c の a、b、c を入力し、Choice 部品で色を指定して「描く」ボタンを押すと、その 2 次曲線のグラフ (の適当な範囲) が画面に現れる。次のものを描いても前のものがなくならないようにするために、1 つのグラフが 1 つのオブジェクトになるようにクラスを定義すること。

4B-1 ストーリーのあるアニメーションを行うプログラムを作れ。アプレットでも独立プログラムでもよい。個々の物体、動き方、1 つひとつの場面 (や暗転) をクラスとして設計すること。

4B-2 CSV データをファイルから読み込み、各種のグラフ (棒グラフ、折れ線グラフ、積み重ねグラフ、等) を切替え表示できるプログラムを作れ。アプレットでも独立プログラムでもよい。1 つの種類のグラフは 1 つのクラス、そのグラフに含まれる折れ線や棒などの部品はまた別のクラスとなるように設計すること。

4C-1 自分の好きな、ただし「4C レベルだと自分が考える」プログラムを設計し製作せよ。アプレットでも独立プログラムでもよい。